# CWE Detail – CWE-23

## Description

The product uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize sequences such as ".." that can resolve to a location that is outside of that directory.

## Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2024-37032: Large language model (LLM) management tool does not  
 validate the format of a digest value (CWE-1287) from a  
 private, untrusted model registry, enabling relative  
 path traversal (CWE-23), a.k.a. Probllama

**•** CVE-2022-45918: Chain: a learning management tool debugger uses external input to locate previous session logs (CWE-73) and does not properly validate the given path (CWE-20), allowing for filesystem path traversal using "../" sequences (CWE-24)

**•** CVE-2019-20916: Python package manager does not correctly restrict the filename specified in a Content-Disposition header, allowing arbitrary file read using path traversal sequences such as "../"

**•** CVE-2022-24877: directory traversal in Go-based Kubernetes operator app allows accessing data from the controller's pod file system via ../ sequences in a yaml file

**•** CVE-2020-4053: a Kubernetes package manager written in Go allows malicious plugins to inject path traversal sequences into a plugin archive ("Zip slip") to copy a file outside the intended directory

**•** CVE-2021-21972: Chain: Cloud computing virtualization platform does not require authentication for upload of a tar format file (CWE-306), then uses .. path traversal sequences (CWE-23) in the file to access unexpected files, as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2019-10743: Go-based archive library allows extraction of files to locations outside of the target folder with "../" path traversal sequences in filenames in a zip file, aka "Zip Slip"

**•** CVE-2002-0298: Server allows remote attackers to cause a denial of service via certain HTTP GET requests containing a %2e%2e (encoded dot-dot), several "/../" sequences, or several "../" in a URI.

**•** CVE-2002-0661: "\" not in denylist for web server, allowing path traversal attacks when the server is run in Windows and other OSes.

**•** CVE-2002-0946: Arbitrary files may be read files via ..\ (dot dot) sequences in an HTTP request.

**•** CVE-2002-1042: Directory traversal vulnerability in search engine for web server allows remote attackers to read arbitrary files via "..\" sequences in queries.

**•** CVE-2002-1209: Directory traversal vulnerability in FTP server allows remote attackers to read arbitrary files via "..\" sequences in a GET request.

**•** CVE-2002-1178: Directory traversal vulnerability in servlet allows remote attackers to execute arbitrary commands via "..\" sequences in an HTTP request.

**•** CVE-2002-1987: Protection mechanism checks for "/.." but doesn't account for Windows-specific "\.." allowing read of arbitrary files.

**•** CVE-2005-2142: Directory traversal vulnerability in FTP server allows remote authenticated attackers to list arbitrary directories via a "\.." sequence in an LS command.

**•** CVE-2002-0160: The administration function in Access Control Server allows remote attackers to read HTML, Java class, and image files outside the web root via a "..\.." sequence in the URL to port 2002.

**•** CVE-2001-0467: "\..." in web server

**•** CVE-2001-0963: "..." in cd command in FTP server

**•** CVE-2001-1193: "..." in cd command in FTP server

**•** CVE-2001-1131: "..." in cd command in FTP server

**•** CVE-2001-0480: read of arbitrary files and directories using GET or CD with "..." in Windows-based FTP server.

**•** CVE-2002-0288: read files using "." and Unicode-encoded "/" or "\" characters in the URL.

**•** CVE-2003-0313: Directory listing of web server using "..."

**•** CVE-2005-1658: Triple dot

**•** CVE-2000-0240: read files via "/........../" in URL

**•** CVE-2000-0773: read files via "...." in web server

**•** CVE-1999-1082: read files via "......" in web server (doubled triple dot?)

**•** CVE-2004-2121: read files via "......" in web server (doubled triple dot?)

**•** CVE-2001-0491: multiple attacks using "..", "...", and "...." in different commands

**•** CVE-2001-0615: "..." or "...." in chat server

**•** CVE-2005-2169: chain: ".../...//" bypasses protection mechanism using regexp's that remove "../" resulting in collapse into an unsafe value "../" (CWE-182) and resultant path traversal.

**•** CVE-2005-0202: ".../....///" bypasses regexp's that remove "./" and "../"

**•** CVE-2004-1670: Mail server allows remote attackers to create arbitrary directories via a ".." or rename arbitrary files via a "....//" in user supplied parameters.

## Related Attack Patterns (CAPEC)

* CAPEC-139
* CAPEC-76

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands — Notes: The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.

**•** Impact: Modify Files or Directories — Notes: The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end of a password file may allow an attacker to bypass authentication.

**•** Impact: Read Files or Directories — Notes: The attacker may be able read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.

**•** Impact: DoS: Crash, Exit, or Restart — Notes: The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the product from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the product.

## Potential Mitigations

**•** Implementation: Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../...//" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string. (Effectiveness: N/A)

**•** Implementation: Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked. Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links (CWE-23, CWE-59). This includes: realpath() in C getCanonicalPath() in Java GetFullPath() in ASP.NET realpath() or abs\_path() in Perl realpath() in PHP (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** A simple way to execute this attack is like this:

**•** While the programmer intends to access files such as "/users/cwe/profiles/alice" or "/users/cwe/profiles/bob", there is no verification of the incoming user parameter. An attacker could provide a string such as:

**•** When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.